CHAPTER 10
BUILDING I/O BRIDGES

In this chapter I assume that you already have a PC peripheral, such as a modem, scanner, or other device that attaches to the PC's RS-232 serial port, parallel port or ISA bus. Your peripheral works fine, but you're considering a USB connection because of the excitement around USB and the reality that these "legacy" connections will be eliminated on future PCs.

I'll present multiple alternatives so you can choose the one that's best for your product.

- First we'll design a USB-to-serial bridge. I've supplied example code for this alternative.
- Then we'll design a high-speed communications peripheral—a 56-Kbps modem—that uses a USB connection.
- We'll implement a design that takes a generic ISA based card and migrates it into a USB peripheral
- Finally we'll look at other standard connections found in the PC industry, such as floppy disk, hard disk, SCSI, and Local Area Network standards, and build bridges from all of them to USB.

Adding USB capability to an existing product is straightforward, and the new product inherits all of the benefits of USB.

DESIGN OF A USB–TO–RS-232 BRIDGE

The RS-232 connection to a PC has proved to be a popular interface choice for peripheral device manufacturers—so popular, in fact, that many users have had to buy more plug-in serial port cards or external switch boxes to accommodate all of the peripherals! The original IBM PC had a single serial port and allowed adding three more. The MS-DOS operating system called these ports COM1, COM2, COM3, and COM4, and most Windows applications also assume that up to four devices exist.

One drawback of the RS-232 connection is that it is point-to-point, which means that only one peripheral device can be connected to one serial port. Multidrop serial port connections, such as RS-422, exist in the industry but have not proved popular in the PC peripheral business because no PC system manufacturer has put an RS-422 connector on their motherboard.

In contrast, USB can be thought of as a multidrop, high-speed connection. Note that USB is NOT a multidrop serial bus like RS-422; it is point-to-point, and hubs are used to enable more connectivity. But a USB solution operates as, and has similar attributes to, a multidrop solution.

Figure 10-1 shows an existing PC peripheral connected to a PC host using an RS-232 connection.

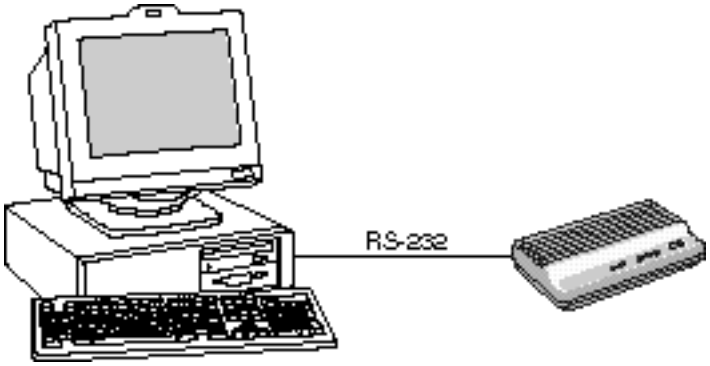


Figure 10-1. Using an RS-232 connection between PC and a peripheral

Using our buttons-and-lights example from Chapter 7, we swap the buttons for an RS-232 receiver and swap the lights for an RS-232 transmitter. Then we modify the firmware to match, and we have thus built a USB–to–RS-232 bridge, or dongle (Figure 10-2).

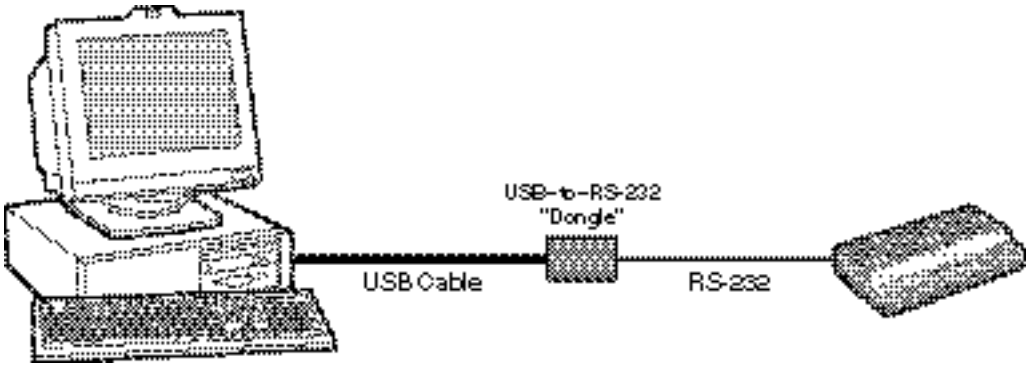


Figure 10-2. Using a USB–to–RS-232 dongle

All of the microcontrollers used in the Chapter 7 examples could easily do this USB–to–RS-232 bridging task. The task is simpler if the microcontroller has an integrated serial port, but the software needed to "bit-bang" a serial port is straightforward (there's an example in the Chapter 10/Serial Port directory on the CD-ROM).

If the serial I/O device has a low data rate, then this converted buttons-and-lights approach is a good solution. If the serial I/O device is a communications device, such as a modem, there's a better way of doing it, and a full USB modem design is implemented later in this chapter. Figure 10-3 shows a representative low-data-rate serial device. It is a security badge system from RFIdeas, and we'll use it as an example of an RS-232 serial device that will have a USB interface added.

Courtesy of RFIdeas, Inc.

Figure 10-3. Representative low-data-rate serial device

The RFIdeas AIR ID product is a proximity-activated identification system. You wear the badge, and the detector is attached to the PC that needs to be protected. When you walk up to the PC, you are automatically recognized and logged on. More important, however, is that when you walk away, you are automatically logged off. This simple security method can prevent unauthorized access to company data.

Looking more closely at the dongle and receiver connection, we find two microcontrollers: one in the bridge and one in the AIR ID receiver (Figure 10-4) (many similar devices would have the same block diagram).

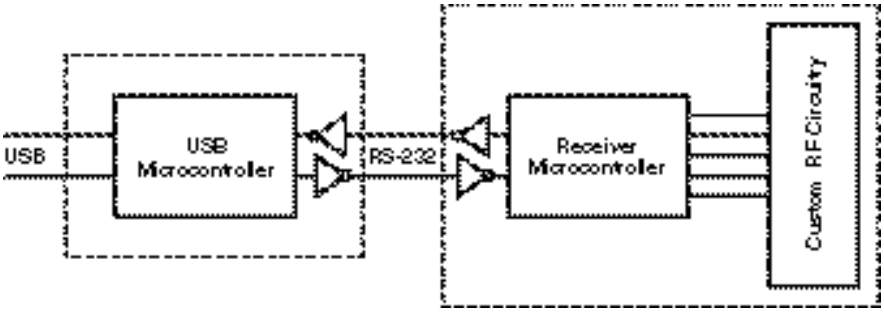


Figure 10-4. Block diagram of dongle plus serial device example

We can make some design optimizations:

- We could use a USB–to–RS-232 bridge as a separate product (several are available; see Appendix A) and offer this as an option with our serial device product. This has the lowest development cost but the highest product cost; it is useful for test-marketing a "USB-based" product.
- We could move the bridge microcontroller inside the unit (Figure 10-5). This also has a low development cost and has the flexibility of including both a serial interface and a USB interface on the product.

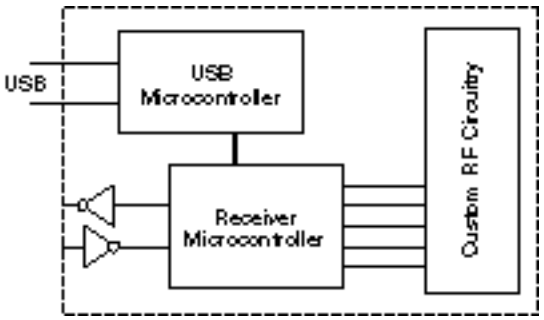
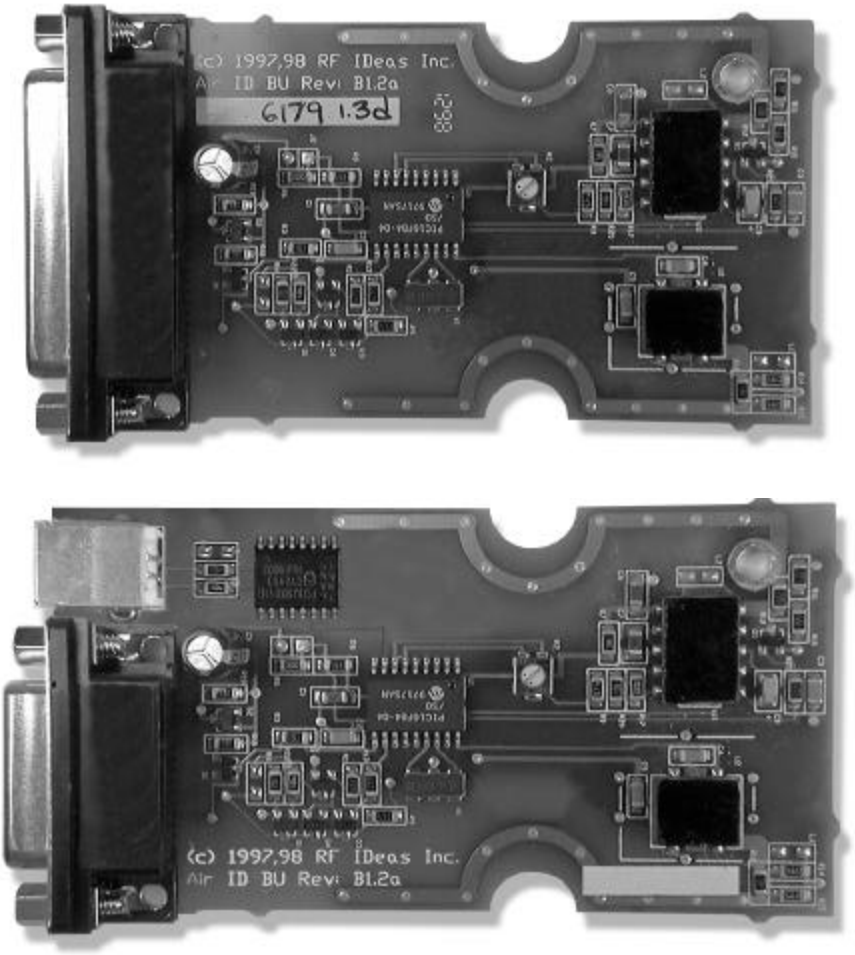


Figure 10-5. Moving the bridge inside the serial product

If our USB product proves to be very successful, we will then want to cost-reduce the dual microcontroller solution shown in Figure 10-5. The ease of this solution depends on which two microcontrollers we need to merge. If the product microcontroller is an MCS51-based microcontroller, the task is manageable, because we have included working examples with this book. If your product uses a different microcontroller, I recommend the I2C-to-USB approach. This approach involves porting the I2C example code to your microcontroller. Figure 10-6 shows an example of a completed product—it includes both a manufacturing option to build RS-232 units (as long as customers still want to buy them) and a USB option for new customers.



Courtesy of RFIdeas, Inc.

Figure 10-6. USB product built from a serial device

Now that we have a USB-connected peripheral device, we can consider adding more features. An RS-232 connection has a data channel in both directions. A USB connection has this data channel but also has a control channel to the peripheral device. Does your peripheral have option switches to set, or does it need to be calibrated, or does it need to be customized at run time? All of these features can be added at no extra hardware product cost. The USB version will be self-identifying and easier for customers to use.

On the other hand, if you are convinced that all that you want is a USB-to-RS232 adaptor then several manufacturers make these as turn-key products and additionally include device drivers that emulate COMx ports (see Appendix A).

The previous section focused on the RS-232 legacy connection on the PC host. The same technique can be used with parallel port devices or game port devices. The USB microcontroller is implemented as a bridge between USB and a parallel connection or a game port connection. You could select a ready-built USB–to–parallel adapter (see Appendix A), or you could integrate the device into your I/O device—similar to our serial port design.

Figure 10-7 shows a clever implementation of a game port device from Microsoft. The microcontroller inside the joystick also supports USB and feeds the D+ and D– signals to two unused pins on the game port connector. An adapter cable allows the joystick to be used in a USB environment.

Courtesy of Microsoft Corporation.

Figure 10-7. Combination game port/USB device

DESIGN OF A SERIAL COMMUNICATIONS PERIPHERAL

Our buttons-and-lights examples were implemented as HID devices so that the software task would be simpler. The HID class was designed for low-data-rate peripherals that typically interact with a person; this class does not support bulk or isochronous transfers. Interacting with another computer can be done at a much higher data rate, and the USB specification includes a Communications Class that has optimizations for this implementation. Let's use the same example of an RS-232 connected communications device, i.e., a modem, that we will migrate to a USB-connected modem.

First, a Look at Communications Standards

Let's look at the USB Communications Class in general and then investigate how USB is having a dramatic impact on the modem industry.

A great number of communications standards exist in the industry, and many of these are evolving to keep pace with new technologies that push the speed of information transfer higher and higher while bringing the cost lower and lower. The USB Communications Class does not invent new protocols or standards, nor does it dictate how existing communications equipment should be used. Rather, it defines an architecture that embraces all existing equipment but with an initial focus on telecommunications services and medium-speed networking services.

The Communications Class defines three classes:

- Communications Device Class—this is a device-level definition that identifies a communications device that has several different types of interfaces.
- Communications Interface Class—defines a general-purpose mechanism that supports all types of communication services. This is the primary interface for device management; it includes the setup and teardown of calls and management of the device's operational parameters.
- Data Interface Class—defines a general-purpose mechanism to support octet data streams using bulk or isochronous transfers. This class is used when another predefined class, such as Audio, is not appropriate.

The Communications Device Class currently outlines the following two categories of devices:

- Telecommunications devices—analog modems, ISDN (Integrated Services Digital Network) terminal adapters, digital telephones, and analog telephones
- Networking devices—ADSL modems (asymmetric digital subscriber line), cable modems, 10BASE-T Ethernet adapters, and Ethernet equivalents

A device will identify itself as a Communications Class device by entering a value of 02 as the Class code value of a device or interface descriptor. Table 10-1 shows several subclasses that are currently defined; many of these have different protocols defined.

Table 10-1. Communication interface subclass codes

Code	Subclass
0	Reserved
1	Direct Line Control Model
2	Abstract Control Model
3	Telephone Control Model
4	Multi-Channel Control Model
5	CAPI Control Model
6	Ethernet Networking Control Model
7	ATM Networking Control Model
8-127	Reserved for Future Use
128-255	Reserved for Vendor-specific Use

As seen in Table 10-1, the list of supported models is extensive. We'll work through a Direct Line Control example. Many other options are available, and the interested reader should refer to "Class Definitions for Communications Devices" in the USB Documents directory on the CD-ROM.

Direct Line Control Modem Example

A Direct Line Control modem is a recent modem implementation that has been enabled by USB. The software is simplified because of advances in modem hardware design. We'll use this hardware implementation for discussion, because it has a direct impact on the software. Figure 10-8 shows a traditional modem connected to USB using the bridge design described earlier in this chapter. Note the large collection of components: We can eliminate many of them to create a lower-cost, yet higher-performance modem.

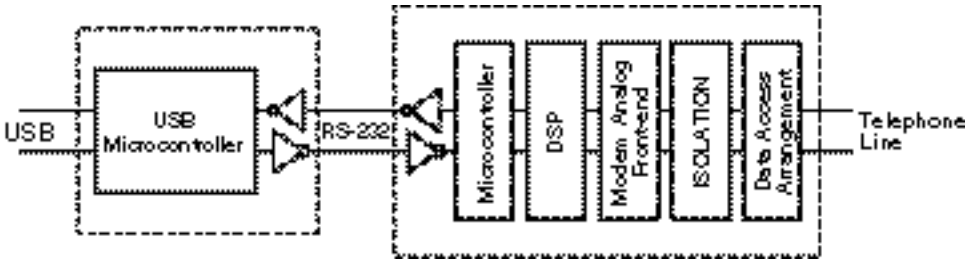


Figure 10-8. Traditional modem with USB–to–RS-232 bridge

Most traditional modems are implemented with four to six major components. These include:

- Microcontroller responsible for interpreting the Hayes modem control codes (the "AT" command set) as well as other housekeeping and speaker driver
- Digital signal processor (DSP) for signal modulation and demodulation
- Analog modulation and demodulation
- Equipment isolation
- Data access arrangement (DAA)

A modem used to be a straightforward device but today includes increasingly sophisticated modulation techniques to squeeze larger amounts of data through a fixed-bandwidth telephone line. Now, what can we start eliminating in our race to keep up with the data flow?

For example, a recently approved scheme, *ITU V.34*, defines a 1000+ point Quadrature Amplitude Modulation (QAM) trellis coding that can support 33 Kbps or 56 Kbps under certain circumstances. Designers no longer use analog methods to create these signals; the arrival of relatively inexpensive digital signal processors (DSPs) in the late 1980s changed this. The Internet has also been a driving force for change as consumers have demanded lower costs for higher speeds. USB is going to create as big a change as single-chip DSPs did.

The DAA is particular to a country and its regulations. The common thread throughout these regulations is signal isolation: The modem, or other equipment, must not be able to inject high voltages or currents into the telephone line. This isolation has been implemented using a transformer, which is bulky, heavy, and less reliable. We'll see a change here, too.

Removing the RS-232 Connection

The first thing to eliminate is the RS-232 connection. The highest speed that a PC serial port can operate at is 115.2 Kbps, and a USB connection will remove this bottleneck. The RS-232 scheme includes two serial signal paths, one in each direction, that are used to transfer in-band commands and data. Commands and status must be intermingled with data because there is only one channel in each direction. USB supports separate command and data channels, which enables out-of-band status reporting—this segregation of command and data channels enables a simplified command structure.

Moving Command Set Interpretation to the PC Host

Of the components found in a traditional modem, the microcontroller task of interpreting the Hayes command set is better handled by the PC host processor. Microsoft includes a Unimodem driver in its communications stack (Figure 10-9) to implement all of these functions. The Microsoft VCOMM port drivers can redirect output originally destined for the serial ports to be transparently passed to the Windows Driver Model (WDM) class driver. The WDM class driver can then direct this output to a USB device. Input from the USB device can be similarly passed back from the WDM class driver, through the VCOMM driver to the Unimodem driver, and back to the applications program. This fundamental partitioning of the logical I/O to the physical I/O that WDM provides gives Windows its "hardware-independent" feature set.

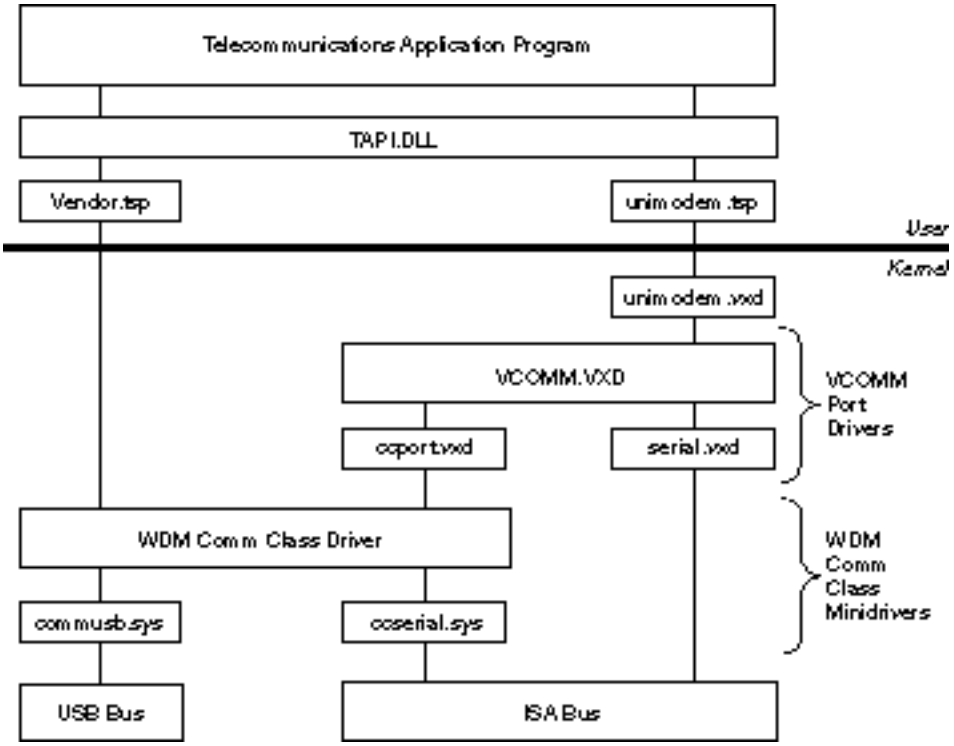


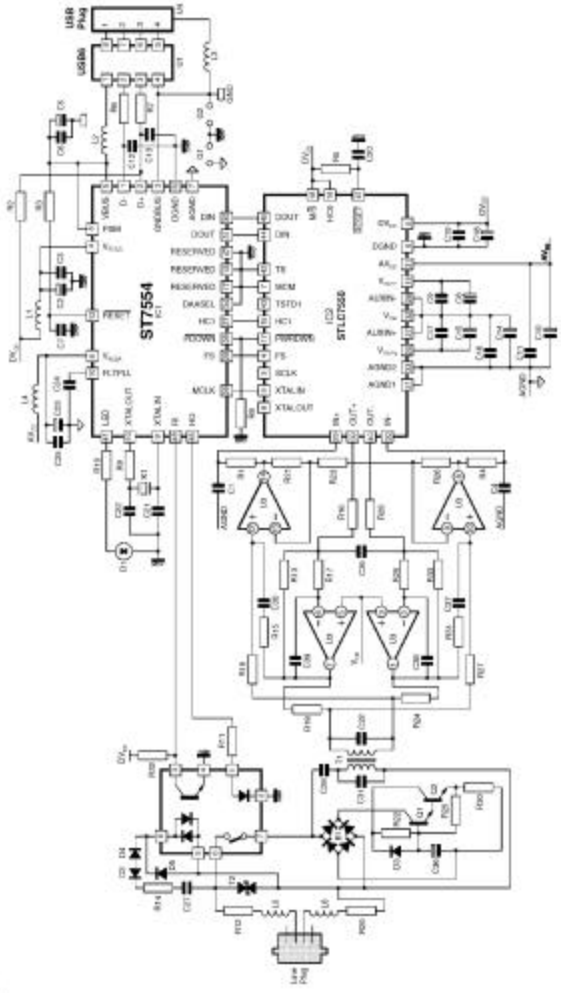
Figure 10-9. Communications stack in the Windows operating system

Removing the DSP Component

The role of the Digital Signal Processor can also be moved to the PC host processor. The modern PC host has ample performance capacity to do significant amounts of real-time signal processing besides the traditional task of supporting user applications. Intel added a set of DSP-like instructions—the MMX instructions—to the original Pentium processor; these instructions are also included in the Pentium II and Pentium III processors. The MMX instructions make short work of the DSP algorithms with the significant side benefit of being a "soft" implementation. The signal-processing algorithm is not fixed as it would be using a physical DSP component. Instead, the algorithm is loaded from disk just like every other PC host program; therefore, it can be easily changed to keep up with the latest, ever-changing communications standards. Lower cost and improved functionality with an obsolescence-proof implementation—the design is going in the right direction!

Can Anything Else be Removed or Simplified?

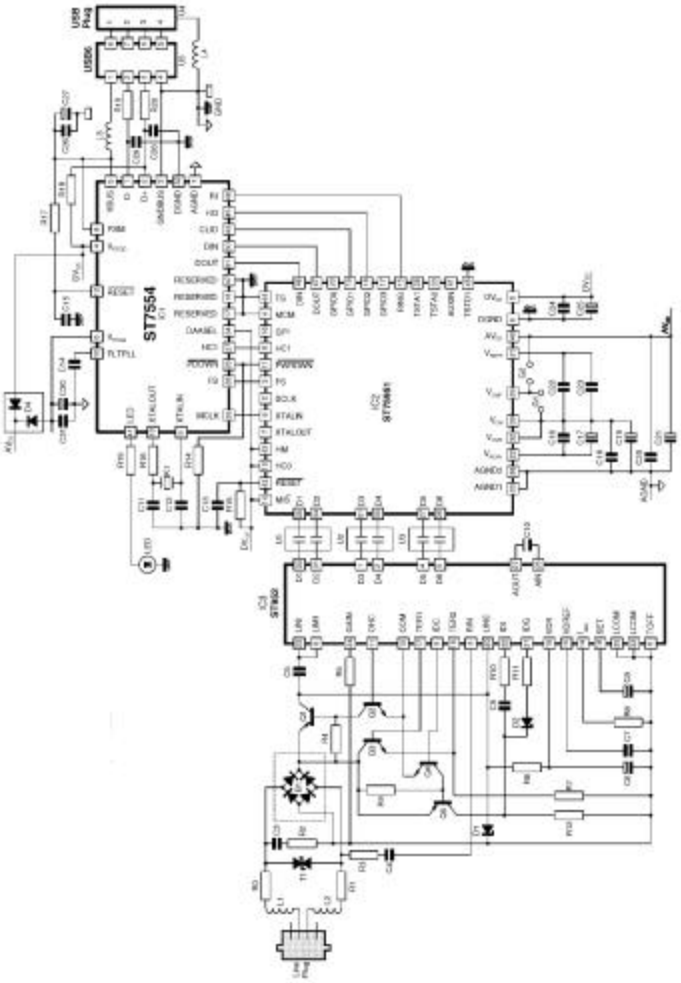
The only elements left in our modem now are the final modulation stages, signal isolation, and a data access arrangement. Figures 10-10 and 10-11 represent how this reduced design can be implemented with and without a transformer (for the full-sized schematics, see the CD-ROM). A reference design guide from STMicroelectronics details these two implementations; the guide is included in the Chapter 10/Modem directory on the CD-ROM.



Courtesy of STMicroelectronics

Figure 10-10. Modem using transformer isolation

(For the full-sized schematic, see the CD-ROM.)



Courtesy of STMicroelectronics

Figure 10-11. Modem using a silicon data access arrangement (DAA)

(For the full-sized schematic, see the CD-ROM.)

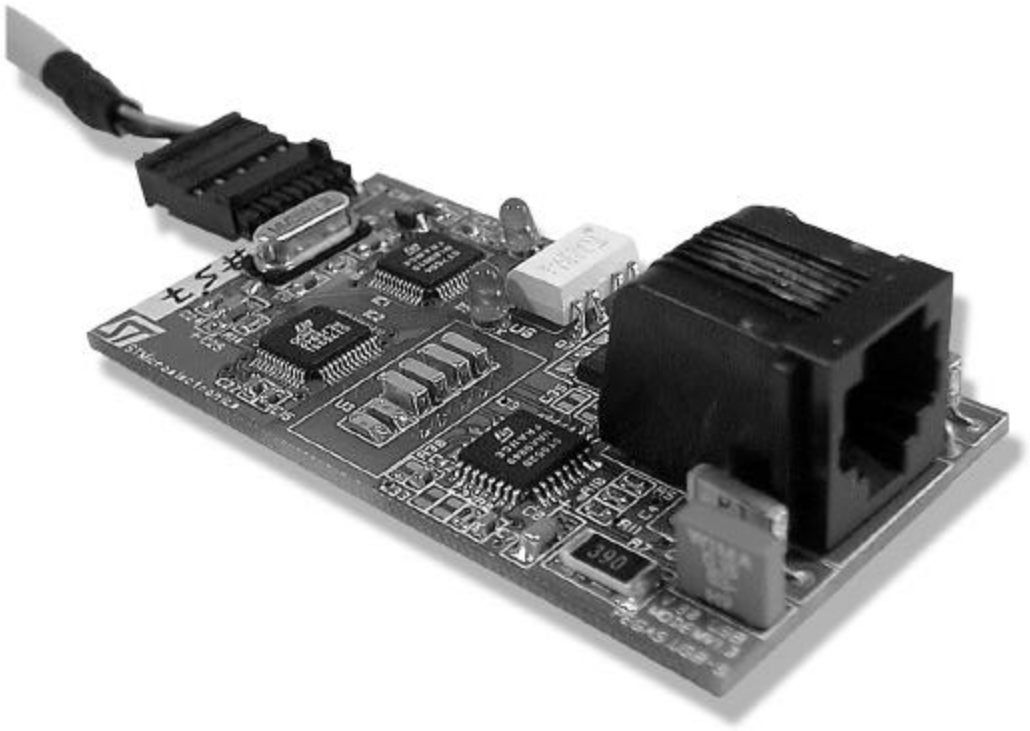
Result of Design Optimization

Figure 10-12 shows a very small transformer-based modem design commercially available from Shark Multimedia Inc.; this 56-Kbps fax/data modem is smaller and lighter than a box of kitchen matches.

Courtesy of Shark Multimedia, Inc.

Figure 10-12. Pocket-sized 56-Kbps modem from Shark Multimedia

Figure 10-13 shows an even smaller and lighter reference design from STMicroelectronics; the transformer has been replaced with a set of high-voltage isolation capacitors.



Courtesy of STMicroelectronics.

Figure 10-13. Transformerless reference design from STMicroelectronics

We have taken almost all the cost out of a high-performance modem by replacing most of the hardware with software on the PC host. The USB architecture facilitates this migration of hardware into software and also enables a simpler direct line model Communications Interface Class, shown in Figure 10-14.

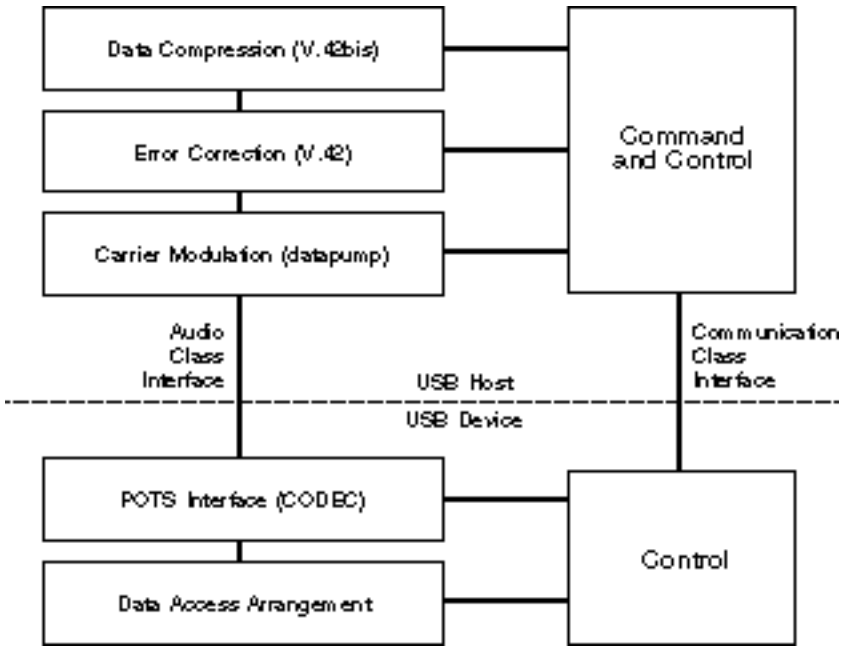


Figure 10-14. Direct line model

The Communications Interface Class uses a minimum of two pipes for control: one for the management function and the other for the notification function. The Data Interface Class also uses two or more pipes. The example in Figure 10-14 used an Audio Class Interface (see Chapter 12) to present the digitally converted data to the PC host. This type of connection would also be useful for a voice-only device such as an answering machine. A basic telephone would also require an HID interface to handle the keypad.

Table 10-2 lists the Requests and Notifications that can be made over the Communications Interface Class. As expected, the requests and notifications are telephone-line specific.

Table 10-2. Direct line model requests and notifications

Request	Code	Description	Req'd/Opt
SET_AUX_LINE_STATE	10h	Request to connect or disconnect secondary jack from POTS circuit or CODEC, depending on hook state.	Optional
SET_HOOK_STATE	11h	Select relay setting for on-hook, off-hook, and caller ID.	Required
PULSE_SETUP	12h	Initiate pulse dialing preparation.	Optional
SEND_PULSE	13h	Request number of make/break cycles to generate.	Optional
SET_PULSE_TIME	14h	Setup value for time of make and break periods when pulse dialing.	Optional
RING_AUX_JACK	15h	Request for a ring signal to be generated on secondary phone jack.	Optional
Notification			
AUX_JACK_HOOK_STATE	08h	Indicates hook state of secondary device plugged into the auxiliary phone jack.	Optional
RING_DETECT	09h	Message to notify host that ring voltage was detected on POTS interface.	Required

The Windows operating system supports all of the Communications Class devices. Today's existing telecommunications software, such as DialTone from Shark Multimedia, can immediately take advantage of these high-performance, low-cost, USB-based modems. New applications can add even more capabilities, such as voice-controlled dialing, using the extra features that the USB infrastructure provides.

MIGRATION FROM ISA

So what's wrong with ISA? Why am I encouraging you to move off of the ISA bus? It boils down to **system performance**—have you noticed how your Internet access slows down while you are printing, how your screen seems to freeze when scanning or accessing a floppy disk? These effects are due to the operation of the ISA bus.

If we compare the architecture of the first PC (Figure 10-15) with that of a modern PC (Figure 10-16), we notice that both have a processor at the top and the ISA bus at the bottom. The original PC contained an 8-MHz Intel 8088 processor; in many PCs today, the processor is a 800-MHz Intel Pentium III processor. Processor performance has increased by over a thousand times while the ISA bus has remained the same.

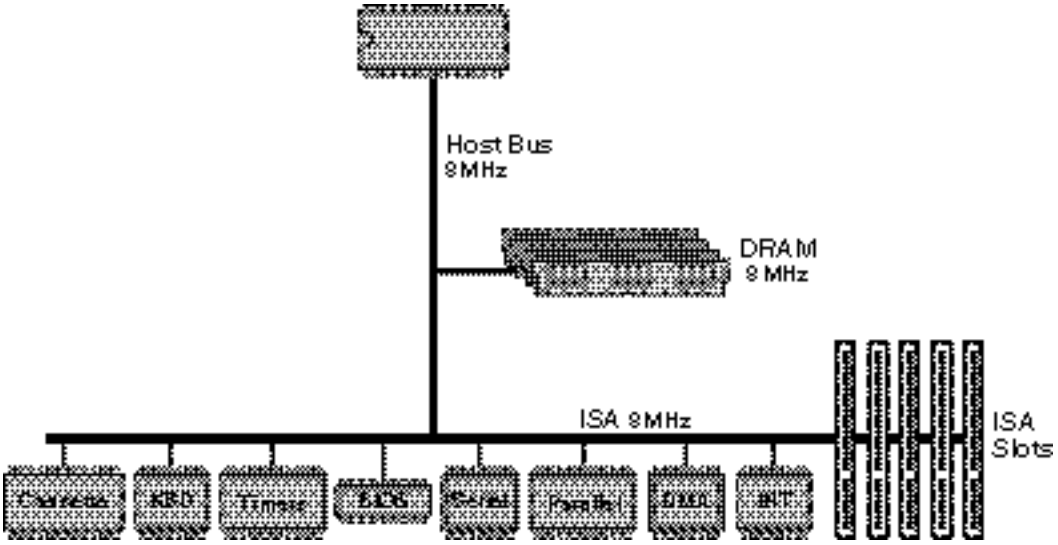


Figure 10-15. Original IBM PC architecture

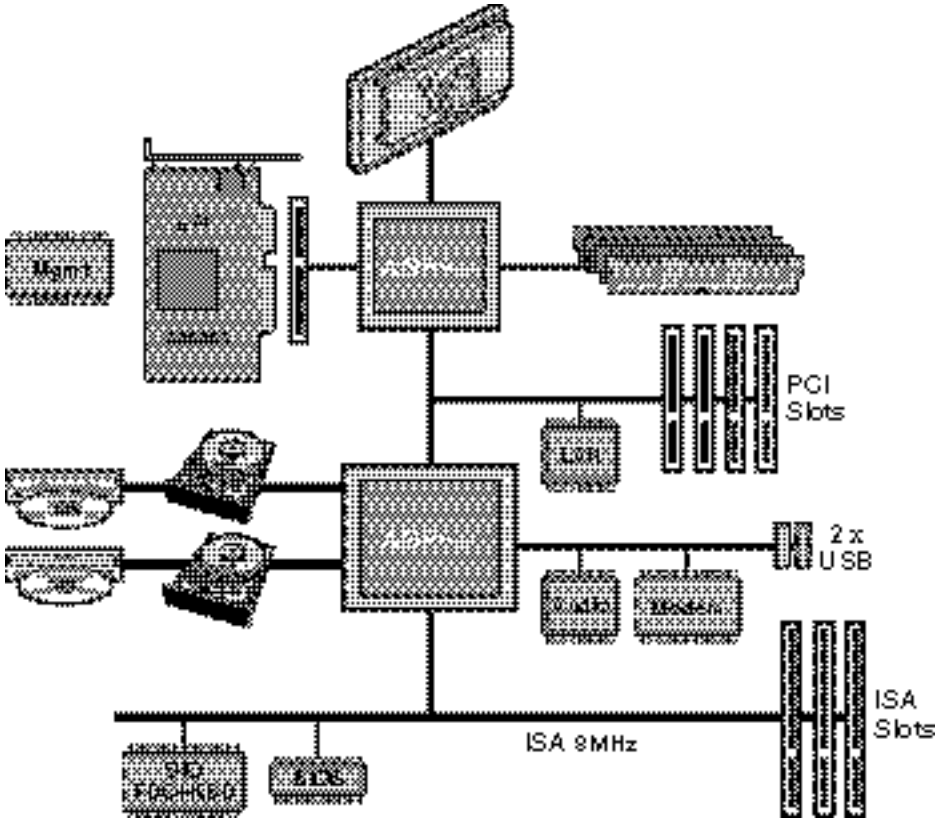


Figure 10-16. Today's PC architecture

Figure 10-17 shows another reason to migrate away from ISA: Tomorrow's PC will not have an ISA bus! The PC 98 and PC 99 System Design Guide documents from Intel and Microsoft have predicted the end of ISA since the end of 1997. PC platforms with ISA-based peripherals will not receive Microsoft's "Designed for Windows" Logo from June 1999 on.

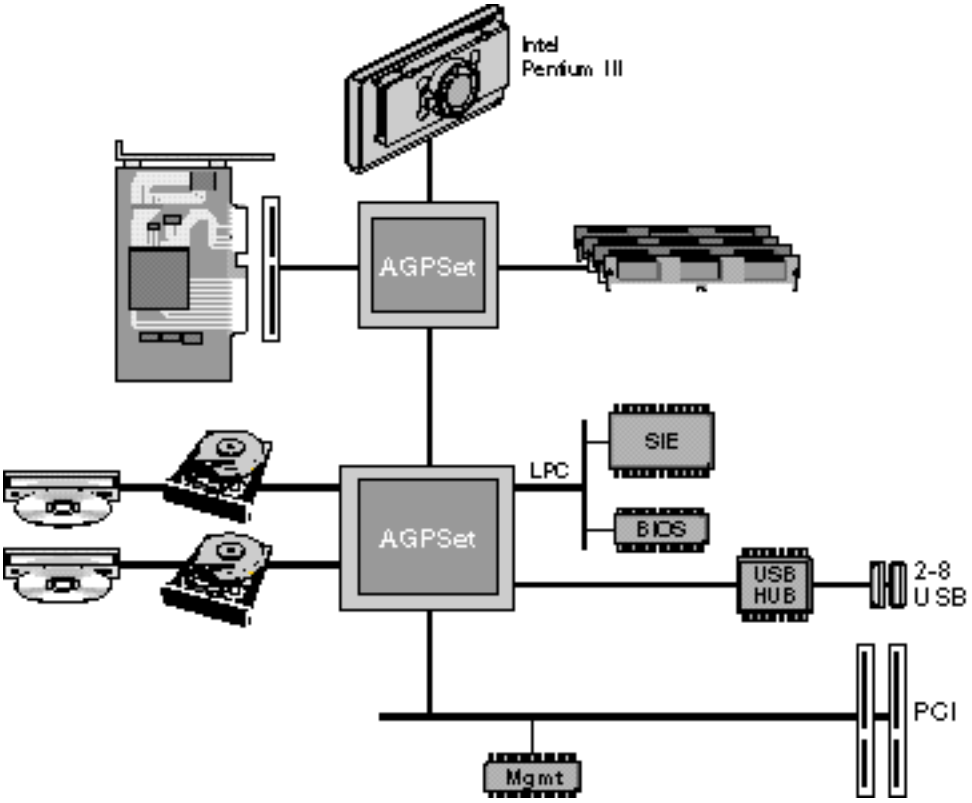


Figure 10-17. Tomorrow's PC architecture

The ISA bus limits growth of the PC platform because of its operation with today's processors and its general lack of modern features such as power management, hot-swap, and configuration management.

IN AND OUT ARE SPECIAL

Fundamental to Intel Architecture microprocessors is the mechanism to access real-world Input/Output. The Intel Architecture instruction set includes specific instructions, IN and OUT, to manipulate data from the real world. Because the timing of a real-world device is not known, the processor waits for an IN or OUT instruction to complete before moving on. In the "old" PC with the 8088 processor, this effect of waiting for completion could be ignored because instruction timings and I/O timings were about the same. Today, if we want optimum system performance, we need to take the IN/OUT semantics into account. The Pentium II processor gains performance by dynamically reordering the instructions it is executing; it also looks ahead in the instruction stream to determine the optimal order in which to execute instructions. The processor will not reorder IN and OUT instructions, because they are tied to events in the real world. While waiting for an IN or OUT instruction to complete, the processor loses the opportunity to optimize program flow.

The real loss in performance is caused by the slow ISA bus. Referring to today's architecture (Figure 10-16), when the processor issues an IN or OUT instruction, it must gain and hold the PCI bus. Once the processor owns the PCI bus, it must then gain and hold the ISA bus; the instruction is completed, and all the buses are now freed. Unfortunately, ISA timing matches the timing of an 8-MHz 8088 processor; so, in the same time period, a 800-MHz Intel Pentium III processor could have executed 800 to 1200 instructions. Retaining the ISA bus in a system costs us the opportunity to increase system performance.

BUILDING AN ISA CARD

Because there was initially no official documentation on how to build an ISA card, many people "reverse-engineered" the requirements from IBM's original design. This worked for a while, until other manufacturers such as Compaq started to build IBM-compatible PCs. The lack of ISA documentation caused minor incompatibilities and resulted in some ISA cards not operating in some machines.

Intel processors support 64 KB of Input/Output address space, but the original PC design included only 10 of the 16 I/O address lines. This resulted in an I/O space of only 1 KB. IBM used some addresses in this space for the system timer, DMA, interrupt controller, and peripherals (serial ports, parallel ports, keyboard, floppy drive, speakers). The I/O addresses were extracted from the BIOS listing by each ISA card builder who located an ISA card in an unused (by IBM) address space. There were, of course, address conflicts; many manufacturers resolved this by making the ISA card address jumper-selectable. But this simply moved the problem to the user.

ISA card manufacturers also used the IBM system interrupts and DMA channels. These too had to be made jumper-selectable, adding confusion for the user. These limited resources were soon used up, and it became burdensome to add more ISA cards to a PC platform.

The inadequacies of ISA were highlighted when the PC industry first started to ship "multimedia upgrade kits." It was difficult to add sound to an early PC, and it required an ample supply of system resources such as interrupt lines and DMA channels. The user was initially reluctant to take the cover off the PC. Would you take the cover off your VCR to "upgrade" it? Tiny jumpers had to be set, ISA cards had to be reinserted into the PC platform, and software had to be installed. It shouldn't surprise us that over 60 percent of these upgrade kits were returned! The upgrade process was too difficult and error-prone for the average consumer to deal with. This was a very valuable (although expensive) lesson for the PC industry. The PC platform had been embraced by the engineering community, but the consumer is a very different customer—to whom even the color and shape of the box are part of the purchase-decision process. Of **paramount** importance is **Ease of Use**.

PLUG AND PLAY ISA

To try to make it easy for a user to add ISA cards to a PC, Microsoft launched a Plug and Play campaign within the ISA card manufacturer community. Many electronics manufacturers collaborated with Microsoft to define mechanisms and standards that would enable ISA cards to be programmable. The operating system would then interrogate the new ISA cards to understand their system resource requirements and to program each card's I/O address, interrupt, and DMA channel to be exclusive.

Unfortunately, the hardware complexity of the resulting Plug and Play implementation far exceeded the functionality of the simpler I/O cards. Manufacturers of the simpler cards ignored the recommendations. Manufacturers of high-functionality cards integrated Plug and Play capability into their bus interface ASICs, so the general ISA situation was improved—but the situation was not ideal.

A modern Plug and Play requirement is power management. With power management, a device powers itself down if it is not being used. The ISA bus cannot support this; in fact, an ISA card can prevent a PC from powering down even when the ISA card is not being used. The ISA bus cannot support "hot-plug" either (hot-plug means that cards are inserted and extracted with the system power still applied). This general lack of features has caused many people to migrate off of ISA already. But you must still have an ISA-based design, because you are reading this section!

To increase the throughput of an ISA system, Compaq Computer proposed and drove a standard extension to ISA called EISA—this extension stretched the bus to 32 bits and added block mode transfers. The Plug and Play scheme for ISA was also embraced and extended. Best of all, however, was a two-tier connector that allowed standard ISA cards to plug in and operate as before while EISA cards would connect to the added signals and thus operate with increased functionality. This extension served the industry well for many years but has now been almost completely replaced by PCI.

USB-TO-ISA BRIDGE

Figure 10-18 shows the strategy that we will adopt – we will move the ISA slots(s) out of the PC host and put them at the other end of a USB cable. The USB microcontroller reconstructs on ISA bus and its firmware manipulates the registers and memory on the ISA card(s). If the ISA card design is simple then the circuitry could be integrated onto the same board as the USB microcontroller.

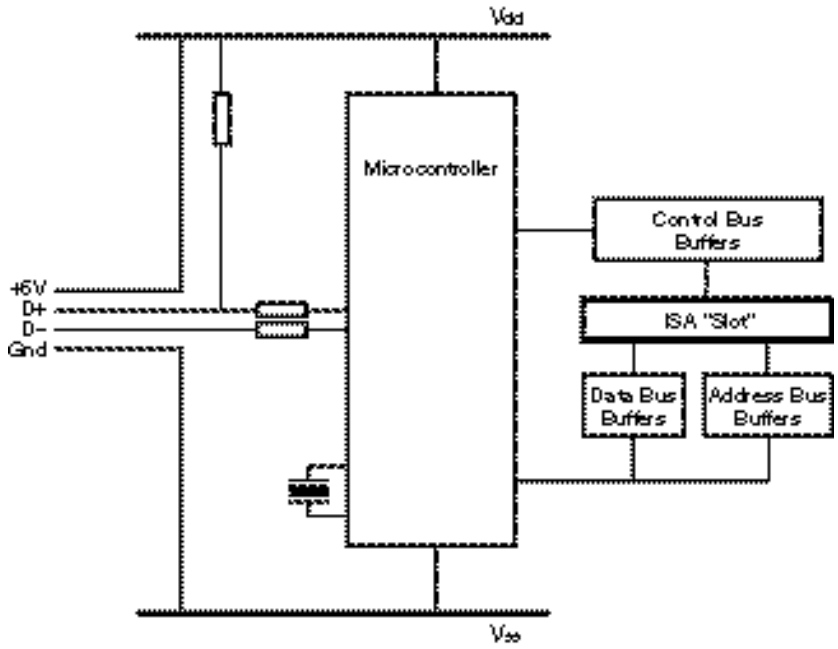


Figure 10-18. ISA card migration

The USB microcontroller in Figure 10-18 re-creates an ISA bus using its I/O ports. The ISA address is generated, and then ISA controls are generated under program control of the microcontroller. The bus timings will **not** be the same as ISA timings, so this introduces some risk. For simple I/O boards, however, the ISA signal timing is not a critical part of the design. Depending on your packaging or other hardware constraints, the ISA interface on your custom I/O board could be replaced with a USB interface. If the total power drawn by the custom I/O is less than 500 mA, then we can even supply the power from USB.

This design method has a huge impact on the software. In my first book I presented a custom VxD that transparently intercepted ISA accesses and

redirected them, via USB, to the ISA slot I/O device (Figure 10-19). The problem with this solution, however, is the I/O latencies involved: ISA I/O accesses take between 100 and 500 usec – USB accesses as described take between 2 and 4 msec, almost 100 times longer. In all practical cases that I investigated this latency was too much to be ignorred and limited the usefulness of this solution. A general solution for ISA migration was not possible since better solutions could be implemented by re-partitioning the applications software – the management and decision making would stay on the PC Host while the real-time aspects would be better moved to the USB microcontroller.

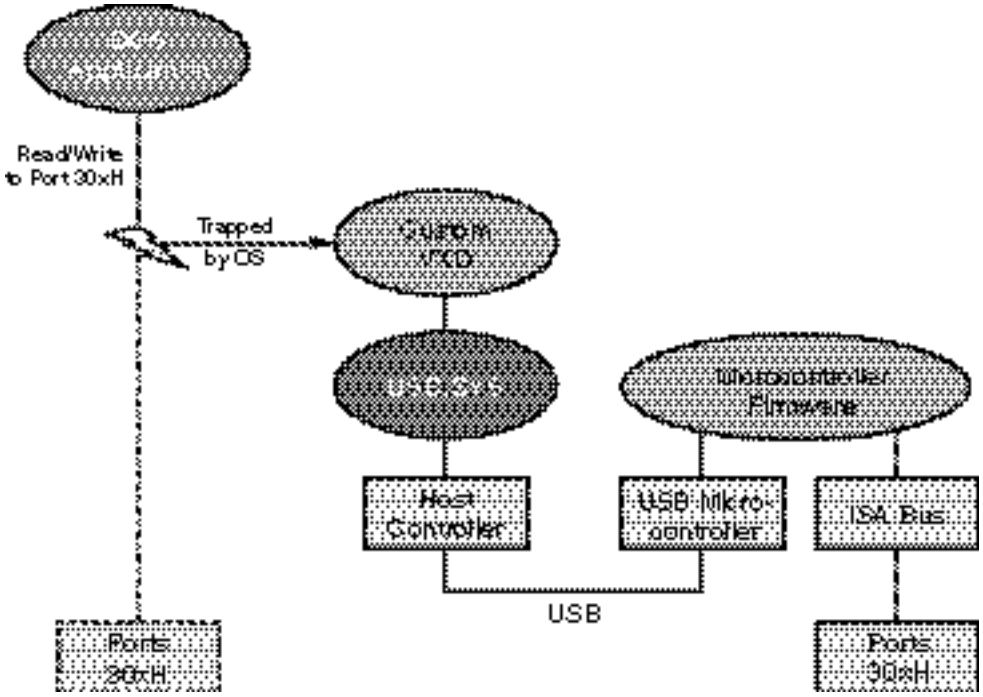


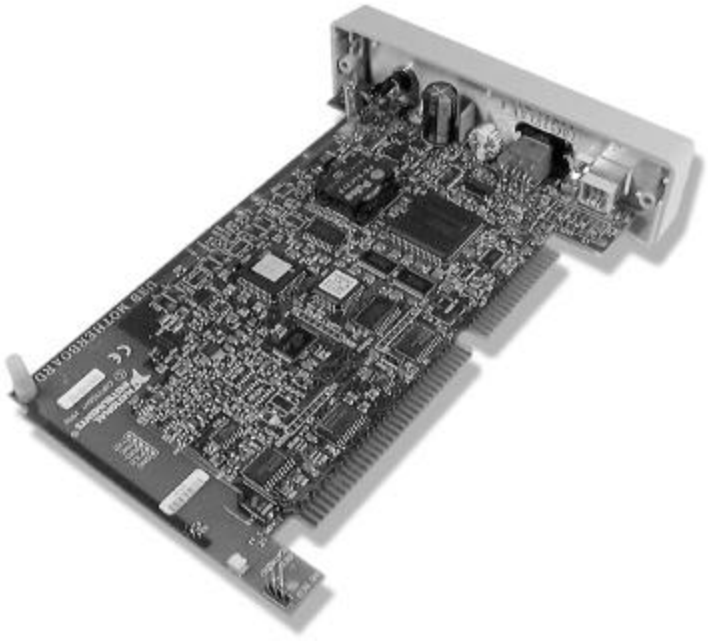
Figure 10-19. Implementing a "hardware redirector"

The "ISA-way" of implementing I/O is to do I/O port polling across the ISA bus; the I/O device is dumb. The "USB-way" of implementing I/O is to pass the I/O device blocks of data and inquire "when you've dealt with that, I can give you some more".

So some software restructuring will be required. Low level bit and byte handling should be managed by the USB microcontroller.

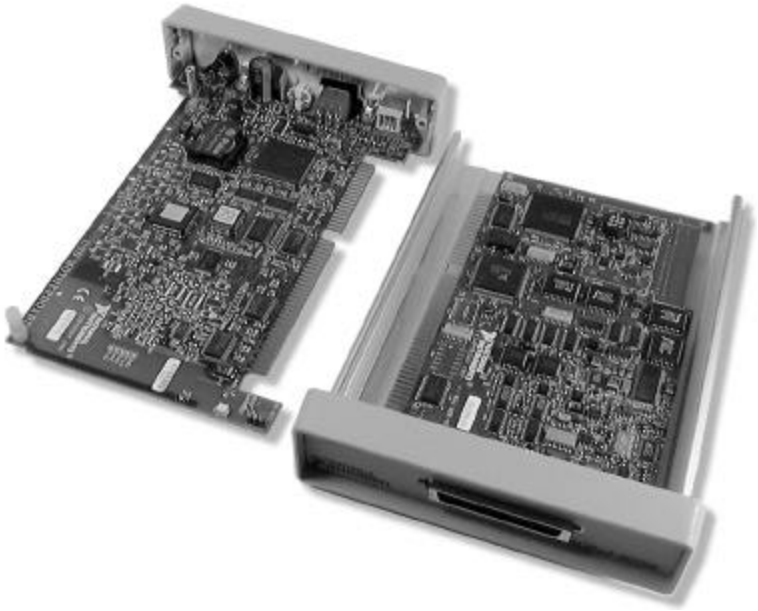
Commercial Example

A very creative solution has been implemented by National Instruments and the idea may apply to your situation. National Instruments has a wide range of individual ISA boards that implement interfaces to analog and digital instrumentation. A custom USB-to-ISA bridge board, shown in Figure 10-20, enabled National Instruments to create an "instant" USB product line. The bridge board, and its companion ISA board, are powered from USB; this allows the pair of boards to be packaged in a convenient, stand-alone enclosure as shown in Figure 10-21.



Courtesy of National Instruments Corp.

Figure 10-20. A custom USB-to-ISA bridge



Courtesy of National Instruments Corp.

Figure 10-21. Creative packaging produces a USB I/O device

Note that National Instruments DO NOT SELL this ISA-USB bridge as a product. They use it only as an internal implementation technique.

PARALLEL DEVICE EXAMPLES

Many parallel interfaces in a PC platform can be replaced by a USB connection. We have already discussed the legacy parallel-printer port, so this section looks at other parallel interfaces both inside and outside of a PC platform chassis.

Floppy Disk Drive

The floppy disk drive is becoming an optional peripheral device in today's PC platform. Most software is now shipped on CD-ROM, and many users are moving to Zip-style (100 MB) or Jaz-style (1 GB) removable media—more about these later. However, it is still useful to have an occasionally connected floppy disk drive that can be shared between multiple users and used as necessary.

Figure 10-22 shows the basic circuitry required to implement an attached floppy drive. The example looks just like our buttons-and-lights example but with a floppy disk controller replacing the switches and LEDs. The microcontroller firmware is much more complicated because of the inherent complexity of the floppy drive controller, but the concept of the design is the same. The I/O device identifies itself as a mass storage device with certain characteristics, and the PC host uses a mass storage device driver to send requests to the I/O device, which implements them via reading and writing from the attached floppy disk drive.

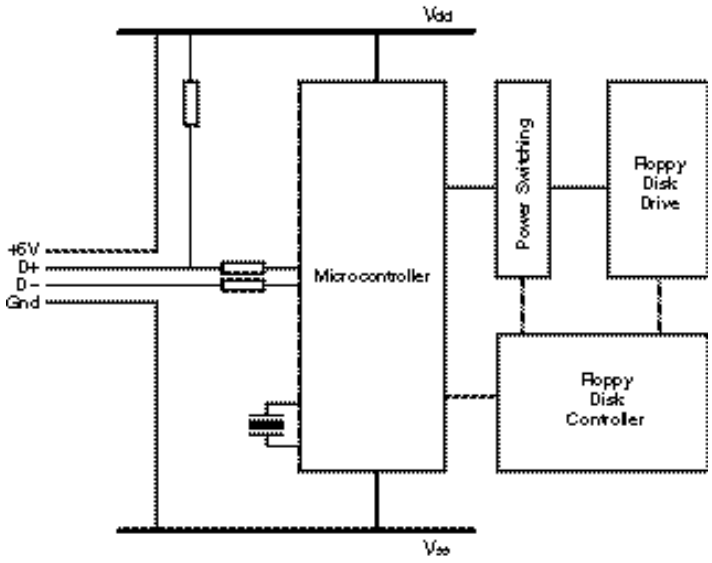


Figure 10-22. Attached foppy disk drive circuitry example

Figure 10-23 shows a commercially available product from Y-E Data. The hardware is partitioned differently, but the block diagram is the same. The unit is powered from the USB cable so no extra connections are required.

Courtesy of Y-E Data.

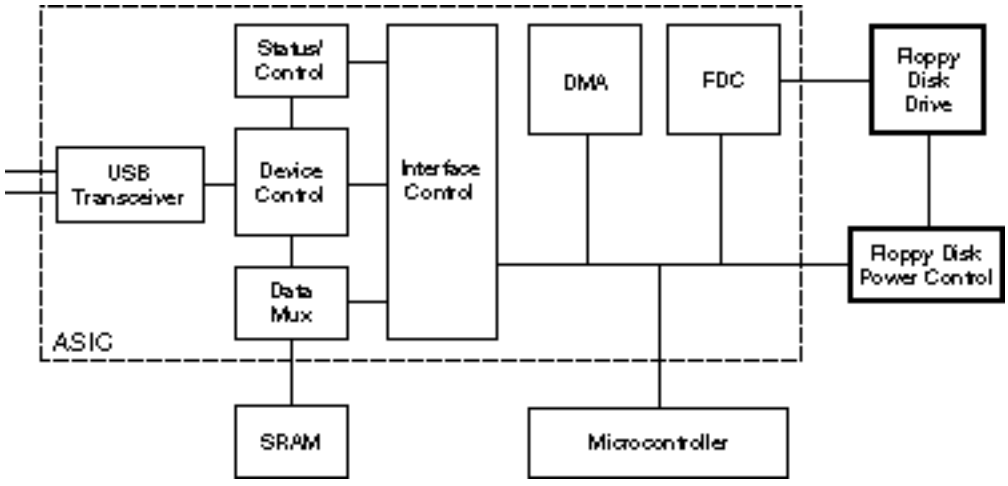


Figure 10-23. Floppy disk drive from of Y-E Data

Hard Disk Drive

The internal hard drive of a PC host has used the IDE (Integrated Drive Electronics) interface for many, many, years. This was pioneered by Compaq Computer Corporation in 1984 and includes intelligence and sector buffering on the disk drive itself. A USB-to-IDE bridge can move the hard drive to the other end of a USB cable, freeing up space inside the PC while, at the same time, creating a portable I/O device. Several USB hard drives available but my favorite is from In-System Design and is shown in Figure 10-24.

Figure 10-24. Portable USB hard drive

Notice that the In-System Design unit does not have a power cord! In-System went the extra engineering mile to create a bus powered device; the unit includes an intelligent power management controller that augments the USB power with rechargeable batteries when needed (ie on initial disk spin up and seeks) and are then trickle charged during normal operation.

In-System Design have already demonstrated a 480Mb/s version of their USB-to-IDE bridge (figure 10-25) which performs faster than traditional internal IDE drivers using standard Mass Storage class drivers.

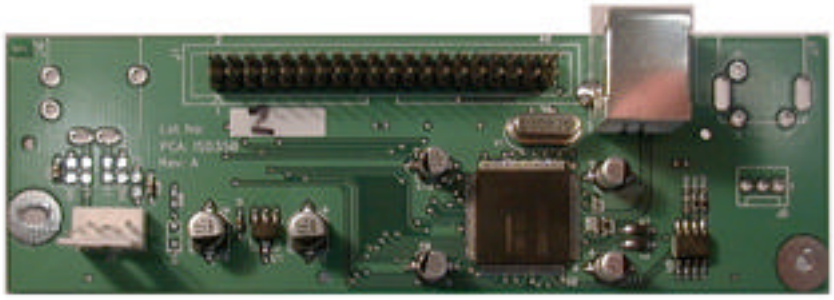


Figure 10-25. 480Mb/s USB-to-IDE bridge

SCSI Devices

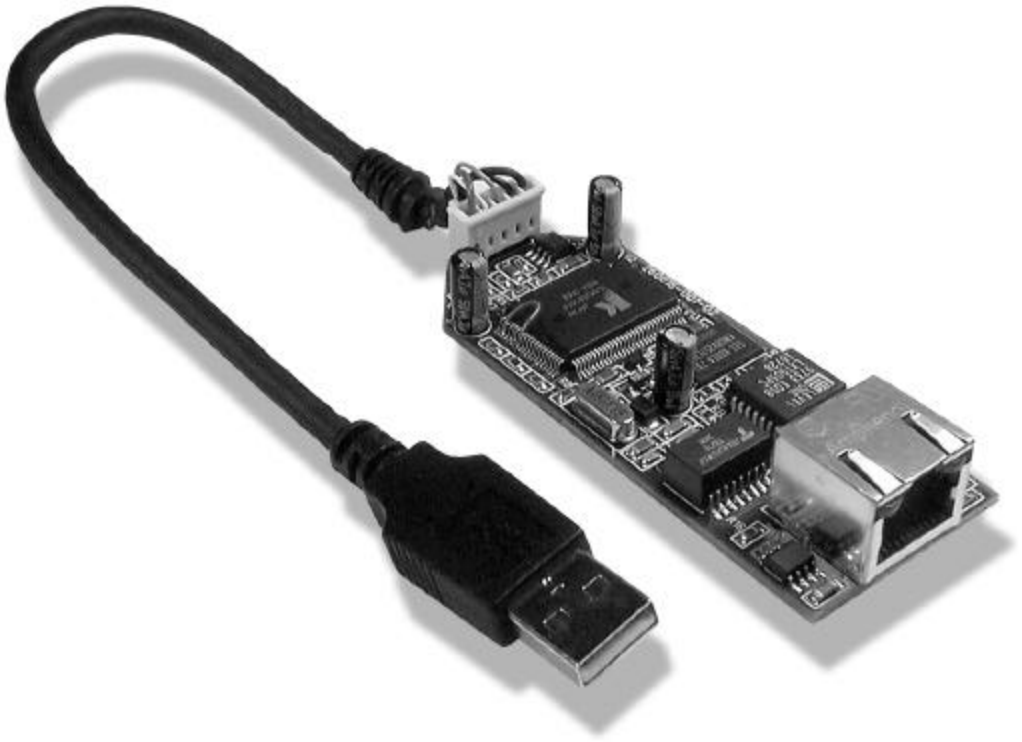
The SCSI parallel connection is a popular interface supporting data rates up to 5 MBps (for a standard interface, both fast and wide interfaces are available with higher data rates). A SCSI connection can be confusing because of the variety of "standard" connectors, the multitude of cables and adapters required, and the fact that signal terminators are required only at the ends of the collection of peripherals. USB is a simpler connection scheme, and manufacturers of popular SCSI devices, such as the SuperDisk drive (left) and Zip drive (right) shown in Figure 10-26, now include a USB option.

Courtesy of Imation (left) and Iomega Corp. (right).

Figure 10-26. USB peripherals derived from SCSI interfaces

USB-to-Networking Bridges

There are many popular local area networking standards and single-chip USB bridges are available for most of them. Figure 10-27 shows a USB-to-Ethernet bridge built around a Kawasaki component – Kawasaki also has 480Mb/s bridge components to Ethernet, Home PNA and PhoneNet.



Courtesy of ADS Technologies.

Figure 10-27. USB bridges to Ethernet

The attractiveness of a USB networking solution is that the PC host need not be opened and a complex PCI card installed. The device driver stack neatly fits into the Windows WDM structure to create a more robust environment.

Bridging to Another PC

USB is designed as a master-slave bus. The PC host is the master and communicates with I/O devices that are slaves. So it is not possible to interconnect two PC hosts with a simple cable. In fact it is dangerous to attempt this since one, or both, of the PC hosts may catch fire. You cannot buy a cable that will allow this (a cable with an A connector on both ends is forbidden) so this risk is minimized.

To bridge between two PC hosts you must use an intervening I/O device that is known by both PC hosts (Figure 10-28). Several manufacturers build these "two-headed" components and supply software that allows the simple networking of two PC hosts.

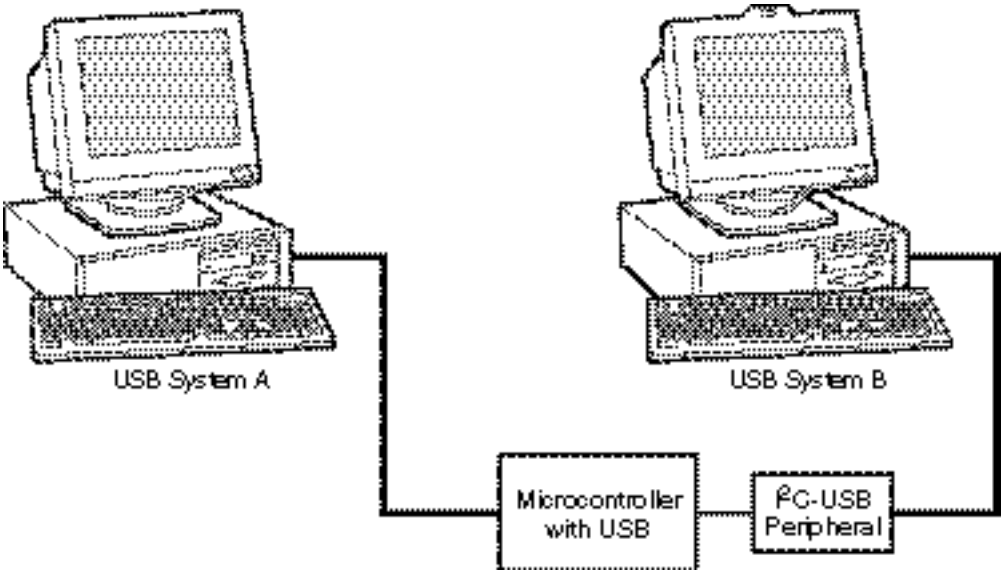


Figure 10-28. PC to PC bridge

It is possible to use multiple of these PC-to-PC bridge components to form a small local area network (Figure 10-29). This configuration is more flexible than Figure 10-29 shows since the PC-to-PC bridge components may be attached to any USB port on any of the systems – this increases configuration flexibility. WARNING: it also increases confusion, you should ensure that each PC host's USB sub-net is isolated from every other PC hosts sub-net.

Figure 10-29. A small LAN using PC-to-PC USB bridges

CHAPTER SUMMARY

I hope this chapter has stimulated thoughts about how almost any device can be simply bridged to USB. Low speed devices use HID class drivers to minimize the amount of software that must be written. Full and high speed devices should be categoried in one of the other supported classes, because this too reduces the amount of software that must be written.

This chapter described the possibility of interfacing a variety of peripheral devices to USB. Having a microcontroller in the peripheral gives us the flexibility to translate the standard reports or packets from the PC host software into custom real-world signals as required by each peripheral. This distributed intelligence approach, a key architectural feature of USB, gives a USB system its flexibility and ease of use.